Описание ClientServerSocketApp

Maven проект ClientServerSocketApp создан как Java-приложение для доступа к серверу технологических данных для интеграции Beremiz и МДЭСПК 2.0 (далее МДЭСПК). Сервер технологических данных (далее SocketServer), реализован как Python-приложение, реализованное с использованием библиотеки Python 2.7, такой выбор версии Python обусловлен требованием по совместимости с ПО Beremiz, куда SocketServer будет впоследствии внедрен по завершению разработки. В целом интеграцию МДЭСПК в Beremiz можно разбить на следующие этапы:

1. первоначально производится разработка SocketServer в виде отдельного Python-приложения, а также разработка API – библиотеки доступа в виде java-класса (далее API доступа) для встраивания в архитектуру МДЭСПК. С целью тестирования данной API подготовлен тестовый пример java-приложения. Описание API приводится непосредственно в проекте (каталог Docs).
2. после того как SocketServer будет протестирован совместно с МДЭСПК, производится интеграция SocketServer в среду Beremiz в виде функционального блока, который будет осуществлять двустороннюю передачу данных от алгоритма управления Beremiz в переменные МДЭСПК (далее тэги). Схема взаимодействия представлена в приложении №1.

# Описание механизма сетевого взаимодействия

За основу механизма сетевого взаимодействия различных по языку программирования вышеуказанных приложений, взят протокол TCP/IP с использованием библиотеки Socket. Приложения могут размещаться как одном компьютере, так и на различных, т. е. иметь распределенную структуру, соответственно для этого в конфигурации приложений, в качестве имени хоста SocketServer, указывается: localhost, либо имя в сети. Кроме этого указывается порт по которому будет осуществляться сетевое взаимодействие. При выборе порта следует учитывать что он имеет соответствующий уровень доступа на чтение и запись для пользователя под которым буду работать данные приложения. По умолчанию настройки следующие:

|  |  |  |
| --- | --- | --- |
| Приложение | Имя Хоста | Порт |
| SocketServer | localhost | 8889 |
| ClientServerSocketApp | localhost | 8889 |

## Алгоритм работы

Сетевое взаимодействие между SocketServer (Сервер) и ClientServerSocketApp (Клиент) осуществляется по следующему алгоритму:

1. Клиент устанавливает связь с Сервером, после чего посылает задание на создание узла, с набором необходимых параметров. Описание узла приведено в приложении №2. Вызов метода:

/\*\*

\* создание узла, с набором необходимых параметров

\* @param id\_node : номер узла

\* @param id\_obj : номер объекта в узле

\* @param d\_value : значение

\* @return

\* @throws IOException

\*/

public int send\_node(int id\_node, int id\_obj, double d\_value)

метод возвращает код ошибки, при этом заполняя ответом структуру узла. Поскольку Сервер еще не интегрирован в Beremiz, то значения меняться не буду. Сервер работает в режиме эхо, понять что сервер успешно отработал можно после анализа кода ответа (i\_code\_answer), который возвращается в ответной телеграмме. Он будет отличаться от первоначального который был в отправляемой на Сервер от Клиента телеграмме. Получить данные узла можно используя следующий метод:

/\*\*

\* получение информации об узле со значением

\* @return str: строка содержащую информацию об узле

\*/

public String getNodeInfo()

1. После того как данные проанализированы закрывается сессия работы клиента

/\*\*

\* метод для завершения сеанса работы клиента

\* @throws IOException

\*/

public void exit\_session()

1. По окончании работы клиента можно завершить работу Сервера, т. к. он работает в многопоточном режиме, то комбинация Ctrl+C не сработает. Для останова используем следующий метод:

/\*\*

\* метод для завершения работы сервера

\* @throws IOException

\*/

public void exit\_server()

Набор реализованных методов можно увидеть в рабочем примере Application

# Установка SocketServer

1. mkdir <targetDir>
2. cd <targetDir>
3. git clone <https://github.com/DVorobiev1968/PyCharm-Socket-Server-2-7.git>
4. если проект уже ранее был клонирован, то обновляемся до последней версии:
   1. git pull
5. убедиться что установлен Python требуемой версии:
   1. python –version
6. должно быть: Python 2.7.2
7. Запуск с параметрами по умолчанию:
   1. **python pysvr.py**
8. Запуск с произвольными параметрами:
   1. **python pysvr.py <host> <port>**

# Установка ClientServerSocketApp

1. mkdir <targetDir>
2. cd <targetDir>
3. git clone https://github.com/DVorobiev1968/ClientServerSocketApp.git
4. если проект уже ранее был клонирован, то обновляемся до последней версии:
   1. git pull
5. mvn clean install



Приложение №2

# Описание узла

Узел представляет совокупность данных включающая в себя целочисленные, строковые, вещественные с плавающей точкой (float) переменные, массивы структур (списки), байтовые массивы. Описание приводится ниже.

Узел различаются по следующим типам:

1. узел для Алгоритмов, т. е. В котором могут храниться только расчетные сигналы от Алгоритмов управления
2. узел технологического сигнала, например от МДЭСПК

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| i\_idNode | идентификатор узла |
| i\_code\_answer | код ответа от узла |
| i\_codeCommand | код команды присваивается в зависимости от протокола работы узла |
| s\_command | описание команды |
| s\_message | строка получаемая из буфера |
| list\_obj | Объект типа список структуры NodeObjInfo |
| **Описание NodeObjInfo** | |
| **Наименование** | **Описание** |
| h\_idObj | идентификатор объекта |
| h\_idSubObj | идентификатор субобъекта |
| i\_typeData | тип данных объекта |
| d\_value | возвращаемое значение |
| i\_check | контрольная сумма |
| b\_message | массив байт отправляемый клиенту |
| b\_obj=bytes() | дополнительный объект для сериализации |

### Описание команд

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Значение** | **описание** |
| CODE\_START | 1 | Множественная загрузка узлов с инициализацией значений тэгов в каждом |
| CODE\_STOP | 2 | Останов Сервера внутри сессии |
| CODE\_SINGLE\_START | 3 | Команда на одиночную работу с конкретным узлом |
| CODE\_SINGLE\_START\_SYNC | 4 | Команда на одиночную работу с конкретным узлом в режиме синхронной передачи данных   1. Сервер принимает телеграмму и анализирует свойство o\_Algoritm.status на признак SET\_ALGORITM\_WAIT, который означает, что узел находится в режиме ожидания расчета алгоритма. Как только данное свойство меняется на: SET\_ALGORITM\_VAL\_OK, SET\_ALGORITM\_VAL\_FAIL, либо инициализируется: CODE\_ALGORITM\_OPERATION, продолжает работу переход на п.2. Иначе дается задержка 20мс, после чего повторный запрос свойства o\_Algoritm.status узла из краткосрочного архива. В случае если изменений в течении 1 минуты не произойдет ожидание завершится. 2. Структура nodeStruct заполняется из содержимого телеграммы CAN, свойство o\_Algoritm.status выставляется признак SET\_ALGORITM\_VAL\_OK 3. Записывается в краткосрочный архив 4. На клиент отправляется телеграмма о выполненной операции, с кодом ответа в i\_codeAnswer |
| CODE\_SINGLE\_START\_ASYNC | 5 | Команда на одиночную работу с конкретным узлом в режиме асинхронной передачи данных |
| CODE\_LIST\_NODES | 10 | Команда на формирования перечня загруженных узлов в краткосрочном архиве |
| CODE\_FIND\_NODES | 11 | Команда на поиск созданного узла |
| CODE\_FIND\_NODES\_SYNC | 12 | Команда на поиск созданного узла в режиме синхронной передачи данных |
| CODE\_LOAD\_FOR\_ALGORITM | 13 | Команда на загрузку данных для использования в ФБ Beremiz. Алгоритм работы заключается в следующем:   1. В краткосрочном архиве отыскивается узел, указанный в телеграмме с командой 2. Если такой узел уже существует, берутся св-ва Объекта и сохраняются без изменений 3. Если узла не существует, то он создается вновь. 4. Свойстве o\_Algoritm.status выставляется признак SET\_ALGORITM\_WAIT |
| CODE\_SAVE\_FOR\_ALGORITM | 14 | Команда на запись данных в алгоритме реализованном в ФБ Beremiz |
| CODE\_EXIT | 20 | Команда завершения текущей сессии работы клиента |
| CODE\_EXIT\_SERVER | 21 | Команда завершения работы Сервера технологических данных |